ARMY RESEARCH LABORATORY

Distributed Interactive Simulation (DIS) Network Manager

Kenneth G. Smith

ARL-TR-780 June 1995

NOTICES

Destroy this report when it is no longer needed. DO NOT return it to the originator.

Additional copies of this report may be obtained from the National Technical Information Service, U.S. Department of Commerce, 5285 Port Royal Road, Springfield, VA 22161.

The findings of this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

The use of trade names or manufacturers' names in this report does not constitute endorsement of any commercial product.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 1995	Summary, 1 - 31 Dec 94

4. TITLE AND SUBTITLE

Distributed Interactive Simulation (DIS) Network Manager

5. FUNDING NUMBERS

PR: 1L162618AH80

6. AUTHOR(S)

Kenneth G. Smith

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

U.S. Army Research Laboratory
ATTN: AMSRL-SC-SS
Aberdeen Proving Ground, MD 21005-5067

8. PERFORMING ORGANIZATION REPORT NUMBER

ARL-TR-780

9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)

10.SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 words)***

Current Department of Defense simulation research centers on real-time, interactive simulation of realistic, complex, "virtual worlds" represent the coalescence of a diverse set of virtual, constructive, and live simulations occurring at various locations throughout the world. This effort is collectively known as Distributed Interactive Simulation (DIS). Its purpose is to allow dissimilar autonomous simulation nodes to interoperate in real-time, interactive, distributed simulations. Inter-node communication is achieved by exchanging DIS protocol data units (PDU). To facilitate the development of simulation application programs that use the DIS protocol, the Simulation Methodology Branch of the Advanced Computational and Informational Sciences Directorate, U.S. Army Research Laboratory (ARL), developed a set of DIS software management programs, collectively referred to as the DIS Network Manager. It employs a client-server based architecture, providing a library of software routines to client application programs for creating, handling, sending, and receiving DIS protocol data units.

14. SUBJECT TERMS

distributed interactive simulation (DIS), protocol data units (PDU)

15. NUMBER OF PAGES

28

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

INTENTIONALLY LEFT BLANK.

PREFACE

Current Department of Defense simulation research centers on real–time, interactive simulation of realistic, complex, "virtual worlds" represent the coalescence of a diverse set of virtual, constructive, and live simulations occurring at various locations throughout the world. This effort is collectively known as Distributed Interactive Simulation (DIS). Its purpose is to allow dissimilar autonomous simulation nodes to interoperate in real–time, interactive, distributed simulations. Inter–node communication is achieved by exchanging DIS protocol data units (PDU). To facilitate the development of simulation application programs that use the DIS protocol, the Simulation Methodology Branch of the Advanced Computational and Informational Sciences Directorate, U.S. Army Research Laboratory (ARL), developed a set of DIS software management programs, collectively referred to as the DIS Network Manager.

The author wishes to acknowledge the Naval Post Graduate school and Dr. David Pratt for providing the software from which an early version of the DIS Network Manager was developed. Though most of the software has been rewritten, the User Datagram Protocol (UDP) networking code is still largely unchanged. Software developers that have also contributed include Holly Ingham and Geoffrey Sauerborn, both from the ARL.

INTENTIONALLY LEFT BLANK.

TABLE OF CONTENTS

INTENTIONALLY LEFT BLANK.

LIST OF FIGURES

INTENTIONALLY LEFT BLANK.

1. INTRODUCTION

Current Department of Defense simulation research centers on real–time, interactive simulation of realistic, complex, "virtual worlds" represent the coalescence of a diverse set of virtual, constructive, and live simulations occurring at various locations throughout the world. This effort is collectively known as Distributed Interactive Simulation (DIS). Its purpose is to allow dissimilar autonomous simulation nodes to interoperate in real–time, interactive, distributed simulations. Each simulation node maintains and conveys the state of one or more simulation entities. Simulation nodes communicate simulation entity states over both local and wide area networks by exchanging standard DIS protocol data units (PDUs).

The Simulation Methodology Branch (SMB) of the Advanced Computational and Informational Sciences Directorate (ACISD), U.S. Army Research Laboratory (ARL), developed a set of DIS software management programs, collectively referred to as the DIS Network Manager. These programs are designed to facilitate the development of software application programs that use the DIS protocol and to explore advancing the DIS protocol itself through the addition of new PDUs and alternate PDU management techniques.

2. GENERAL

The DIS Network Manager provides application programs with a programmatic interface to a DIS network. In this report, a DIS network is defined to be any local area network (LAN) on which DIS application programs[1] are operating. The DIS Network Manager comprises two functional components. The first, called *dis_mgr,* establishes and manages a connection to a DIS network. The second component, called *libdis*, is a software library that works in conjunction with the *dis_mgr* to facilitate constructing, sending, receiving, and interpreting DIS PDUs.

2.1 DIS

DIS operates under the notion that a diverse and large number of simulated entities may participate and interact in real time during a single simulation exercise. Successful interaction depends on each simulated entity's physical state and actions being communicated to all other simulated entities in a timely, succinct, and consistent fashion. DIS provides a protocol that describes the philosophy and requisite information needed to simulate entities in such an environment.

Successful DIS simulation exercises are dependent in part on the unique identification of each simulated entity. Simulated entities in DIS are uniquely identified by site, host, and entity numbers. Each physical site participating in an exercise is identified by a unique number. Further, each computer host at a site is identified by a unique number. Lastly, each entity that is being simulated by a computer host is assigned a number unique to that host.

Communication of entity states and actions is accomplished by using predefined DIS PDUs. PDUs convey the current state of a simulated entity or a simulation event involving one or more entities, such as a collision. Within the PDU, details concerning the state or event are specified via enumeration and bit-encoded values.

---

[1] The term "DIS applications"or "DIS application programs" refers to programs that use the DIS communications protocol.

DIS exercises are conducted over both local and wide area networks. They consist of simulation hosts running applications that simulate one or more entities. Each host is responsible for transmitting the absolute truth about the entity(ies) it is simulating to all other hosts in an exercise. It is similarly responsible for determining how its simulated entity(ies) are perceived and affected by incoming PDUs (DIS 1994).

2.2 DIS Manager

The *dis_mgr* is a software program that provides network communications services for application programs. It uses user datagram protocol (UDP) broadcast to transmit outgoing PDUs over the DIS network to other simulation hosts. It listens to the network and receives all incoming PDUs that other simulation hosts have transmitted.

The *dis_mgr* can communicate with other *dis_mgrs* or directly with other DIS applications that are not designed to use a *dis_mgr*. For DIS applications to exchange PDUs with other applications, the only networking requirement is that all applications transmit and receive PDUs over the same UDP ports. While the *dis_mgr* uses default UDP ports for sending and receiving PDUs, the port numbers can be changed through command–line arguments. This allows for the concurrent development and testing of multiple DIS applications on the same LAN without interference from other DIS applications or exercises.

The DIS Network Manager departs from the notion of a single host computer running a single simulation application. Rather, it is designed to permit multiple simulation applications to run from a single host computer. Due to the inherent nature of UDP to only permit a single communications channel per host computer, a client–server architecture is employed by the *dis_mgr* to enable multiple applications to operate on, and transmit their PDUs from, a single host computer.

The client–server protocol implemented between the *dis_mgr* and application programs uses UNIX transport control protocol (TCP) internet protocol (IP) to exchange information. Client programs connect to the *dis_mgr* to use its send and receive services. The *dis_mgr* establishes a UDP connection to the DIS network. It takes outgoing PDUs from its client programs and transmits them over the DIS network. Incoming PDUs that the *dis_mgr* receives from the DIS network are passed back to its application programs (see Figure 1).

An interesting feature of this architecture is that client programs who send PDUs will in turn receive them from the *dis_mgr* as incoming PDUs. This can facilitate an application's processing of entities as the DIS protocol requires that applications maintain two states for each entity in their responsibility: 1) the high fidelity absolute state of an entity as known only by the application, and 2) the "ground truth" state of an entity as it is presented to and represented by all applications participating in the simulation exercise. Receipt of its own PDUs enables an application to more easily maintain the "ground truth" state of its own entities.

Default configuration of the *dis_mgr* permits from one to four client programs to utilize the services of a single *dis_mgr*. It can be configured through a command–line argument to handle up to eight clients. This number was arbitrarily chosen to impose a practical limit on the number of clients for any one *dis_mgr*. A relatively small number was chosen because of the DIS entity identification scheme that requires entities be assigned a unique number between 0–65,535 for each simulation host. If more than one simulation application is running on a single host, then entity id numbers must be unique across all applications on that host. The DIS Network Manager ensures such uniqueness by assigning a range of entity id numbers to each

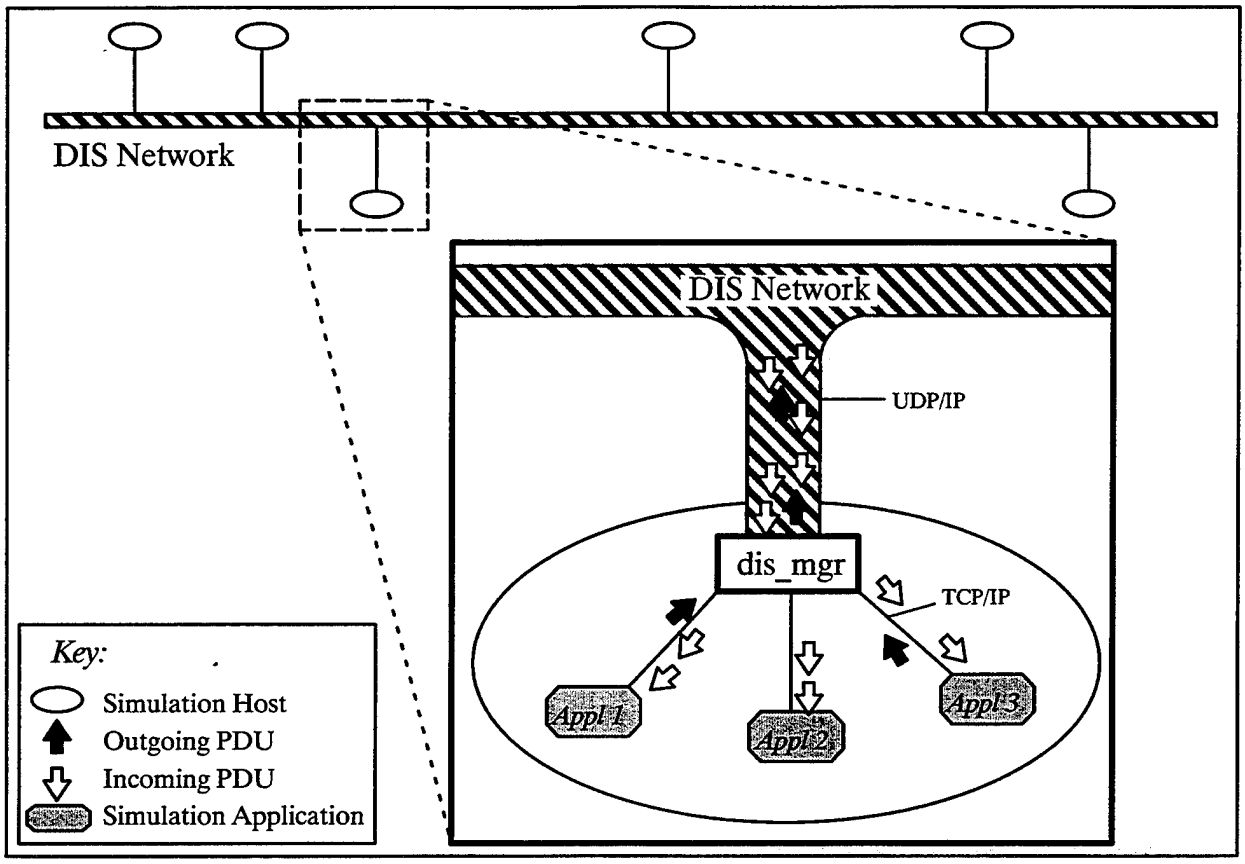

Figure 1. *dis_mgr* Network Connections

application will be assigned the range 0–16,383 to use for entity id numbers. The second application will use numbers in the range 16,384–32,767, etc.

Another service offered by the *dis_mgr* is PDU filtering. PDUs are filtered at the PDU level. That is, they are filtered by PDU type. A client application program can instruct the *dis_mgr* to not send it any PDUs of a specified type, such as FirePDU or ServiceRequestPDU. Thus, applications will only receive those types of PDUs that they specify. This prevents an application program from having to process PDUs that it can not or does not want to handle.

Lastly, the *dis_mgr* queues incoming PDUs for each client application program until the client requests them. The philosophy of the DIS Network Manager is that client application programs will process PDUs as fast as they are able. When applications are ready to receive the next incoming PDU, they request it from the *dis_mgr*. This scheme simplifies the design of applications since they do not have to be developed to handle incoming PDUs asynchronously.

2.3 DIS Library

The second component of the DIS Network Manager, *libdis*, is a library, or collection of software routines, for use by application programs utilizing the *dis_mgr*. It provides a consistent, easy–to–use means of creating, interpreting, sending, and receiving PDUs. It also handles the TCP/IP based client–server communication between application programs and the *dis_mgr*.

Application programs using the services of the *dis_mgr* first connect to it, via the library routine called dis_open(). This routine opens a TCP/IP–based connection to the *dis_mgr* through which all future communication will occur. In response, the *dis_mgr* sends a message back to the application program containing the protocol version, exercise identification number, and range of available entity identification numbers for use by that application when creating entities.

For filtering PDUs, two routines are available to application programs: dis_register_pdu() and dis_ignore_pdu(). When application programs first connect to the *dis_mgr,* they default to receive no incoming PDUs from the *dis_mgr*. They must register with the *dis_mgr* those PDUs that they wish to receive. This is done by listing the desired PDU types via the dis_register_pdu() routine. This ensures that application programs will not receive unexpected PDU types. The dis_ignore_pdu() routine allows application programs to stop receiving PDU types that were previously registered for receipt.

The library provides a set of routines whose names take the form of get_*PDUtype*PDU, where *PDUtype* is the name of a type of PDU. For example, to create an EntityState PDU the routine get_EntityStatePDU() would be used. These routines create C data structures that reflect the fields in the PDU. The "protocol header" field of the newly created PDU is filled by the *libdis* PDU creation routines. It is the application programs responsibility to fill all of the remaining fields in a PDU with appropriate values.

Application programs send PDUs via the dis_send() routine. This routine takes as its argument a pointer to a C–structured PDU as created using one of the get_*PDUtype*PDU() routines. These can be complex structures containing many smaller structures and even linked lists of other structures. The dis_send() library routine compresses the C–structured PDU into a single contiguous string of bytes and fills in the "length" field of the PDU protocol header before sending the PDU to the *dis_mgr* for transmission over the DIS network.

A complementary routine to dis_send() is dis_read(). This routine is used by application programs to receive the next incoming PDU from the *dis_mgr*. PDUs from the *dis_mgr* are received as a single contiguous string of bytes. dis_read() converts those bytes into an appropriate C–structured PDU for easier use by the application program. Additionally, dis_read() returns an integer that identifies the type of PDU that was received.

To assist in debugging DIS application programs, *libdis* provides the capability to print PDUs. The print routines, print_*PDUtype*(), print C–structured PDUs in ASCII and identify the respective data type of each PDU field. The DIS standards specify the data type of each field in a PDU by length and type. For example, the "protocol version" field of PDUs is defined to be an 8–bit unsigned integer. Appendix A contains output generated by the print_EntityState() routine for a sample EntityState PDU.

2.4 Other Features and Utility Programs

A sample client application program was created that illustrates a DIS application program designed to use the DIS Network Manager. This program demonstrates the rudiments of an application program connecting to the *dis_mgr*, registering to receive PDUs, creating a PDU, sending the PDU, and printing any incoming PDUs.

A second, more sophisticated application program was created that provides an X/Motif–based graphical user interface for creating sample PDUs that can be sent out over the DIS network. This program, called *clientX*, offers a pull-down menu of all of the available PDU types

that a user may send along with a popup window for setting the number of PDUs to send. A "Send" button is provided that, when pushed, will create a sample PDU for each type and number of selected PDUs in the menu and send them to the *dis_mgr* for transmission across the DIS network. More than one type of PDU can be selected to be sent at a time.

Functions are provided in the *libdis* library to enable the logging of PDUs. PDUs can be logged with an optional time–stamped header, by direction (incoming, outgoing, or both), and in one of three formats: ASCII, binary, or ASCII–binary. The optional header precedes each logged PDU in a log file and contains a time stamp, the PDU type, and "INCOMING" or "OUTGOING" designation to indicate whether the logged PDU was sent and/or received by the the logging program. ASCII formatted PDUs utilize the print_*PDUtype*() routines in logging PDUs to a file. Binary formatted PDUs are logged in straight binary. The ASCII–binary format logs PDUs by representing a PDU's binary format using ASCII "ones" and "zeros." This presents the actual binary bit stream for a PDU in a human readable form. It requires considerably less space than straight ASCII and is useful when debugging PDUs to examine the actual bit values of a PDU. Appendix B contains sample PDUs logged in both the straight ASCII format and ASCII–binary format.

Both the *dis_mgr* and *clientX* programs are capable of logging PDUs. Logging via the *dis_mgr* is enabled through command line arguments. The *clientX* program has a popup window through which logging can be turned on or off and the various combinations of header, PDU direction, and format can be specified. Using the *dis_mgr* to log PDUs allows logging of all PDUs on a DIS network or simply those PDUs that originate from the dis_mgr's clients. Logging PDUs at the client program level allows for the logging of all PDUs sent and/or received by a particular DIS application.

Complementing the logging capability is a set of utility programs for converting between the straight binary and the ASCII–binary log formats. Additionally, a program is provided to replay logged PDUs. This program plays back time–stamped binary formatted log files. It replays the PDUs at the original rate at which they were sent.

3. FUTURE DEVELOPMENTS

One of the merits of the in–house developed DIS Network Manager is the control over its development and operation. It readily supports the development and testing of new DIS PDUs. Similarly, alternative schemes for exploring more efficient use of the DIS network can be tested. Filtering PDUs at a level above application programs alleviates the application programs of this burden, freeing them to do more processing of their designed tasks.

A more sophisticated PDU filtering scheme is under consideration as a future enhancement to the DIS Network Manager. Currently, PDUs are filtered by type. Other schemes could include filtering based on the source, time, and entity location. In an experiment involving "live" entities, it may be desirable for other live entities to ignore their PDUs if they are redundant. For instance, in an exercise involving live and simulated tanks, the real tanks don't need to receive EntityState PDUs about other real tanks that they can physically see. PDUs could be filtered if they are outdated due to slow or congested networks. PDUs could also be filtered based on their relative impact on the receiving entity. In other words, if an incoming PDU will have no immediate or future impact on the receiving entity then it can be filtered out by the *dis_mgr*.

Another enhancement is a more transparent method of generating entity identification numbers. Entity identification numbers are supposed to be unique within each simulation host. Since the *dis_mgr* acts as a simulation host, but comprises potentially more than one simulation application, all entity identification numbers should be generated by the *dis_mgr*, not each of its client application programs as is currently done.

Future versions of the DIS Network Manager will use an interrupt driven scheme to notify client application programs of the arrival of PDUs. This will alleviate the application programs from having to continually poll the *dis_mgr* for incoming PDUs. The *libdis* library will generate a separate process for each client program that will notify the client program when a PDU has arrived. In this way, client programs will not need to regularly and frequently call dis_read() to check for incoming PDUs. Instead, they will only need to call it when they have been signaled that a PDU is pending.

4. CONCLUSION

The DIS Network Manager was developed to facilitate the creation of applications programs designed to operate in a DIS environment. It provides a DIS network level interface through which application programs can easily send and receive DIS PDUs. Built–in routines give application programs easy control over PDU creation and flow control. The DIS Network Manager is flexible and can be easily expanded to support experimentation with new and altered PDU types and network communication control. Enhancements are currently underway to provide improved logging, playback, analysis, and interactive generation of PDUs.

5. REFERENCES

DIS Steering Committee "The DIS Vision: A Map to the Future of Distributed Simulation." IST–SP–94–01, Institute for Simulation and Training, Orlando, FL, May 1994.

INTENTIONALLY LEFT BLANK.

APPENDIX A:

SAMPLE OUTPUT

INTENTIONALLY LEFT BLANK.

Sample Output from print_EntityStatePDU() Routine

The *libdis* library contains simple debugging routines that print DIS PDUs in ASCII. Below is an example of the output for an EntityStatePDU. The values for the fields in the EntityStatePDU may not be correct or valid as this is just sample data used to test the routine. The version of the DIS protocol being used is 2.0.2.

```
->Entity State PDU:
{
        protocol_header = {
                protocol_version = 2 (8 bit unsigned int)
                exercise_ident = 1 (8 bit unsigned int)
                pdutype = 1 (8 bit enum)
                length = 0 (8 bit unsigned int)
        };
        entity_id = {
                site = 37 (16 bit unsigned int)
                host = 63 (16 bit unsigned int)
                entity_id = 1 (16 bit unsigned int)
        };
        force_id = 2 (8 bit enum)
        entity_type = {
                entity_kind = 0 (8 bit enum)
                domain = 0 (8 bit enum)
                country = 0 (16 bit enum)
                category = 0 (8 bit enum)
                subcategory = 0 (8 bit enum)
                specific = 4 (8 bit enum)
                extra = 0 (8 bit enum)
        };
        alt_entity_type = {
                entity_kind = 0 (8 bit enum)
                domain = 0 (8 bit enum)
                country = 0 (16 bit enum)
                category = 0 (8 bit enum)
                subcategory = 0 (8 bit enum)
                specific = 0 (8 bit enum)
                extra = 0 (8 bit enum)
        };
        time_stamp = 0 (32 bit unsigned int)
        entity_location = {
                x = 1.000000 (64 bit float)
                y = 2.000000 (64 bit float)
                z = 3.000000 (64 bit float)
        };
        entity_linear_velocity = {
                x = 0.000000 (32 bit float)
```

```
                y = 0.000000 (32 bit float)
                z = 0.000000 (32 bit float)
        };
        entity_orientation = {
                psi = 0.000000 (32 bit float)
                theta = 0.000000 (32 bit float)
                phi = 0.000000 (32 bit float)
        };
        dead_reckon_params = {
                algorithm = 0 (8 bit enum)
                linear_accel = {
                        x = 0.000000 (32 bit float)
                        y = 0.000000 (32 bit float)
                        z = 0.000000 (32 bit float)
                };
                angular_velocity = {
                        x = 0.000000 (32 bit float)
                        y = 0.000000 (32 bit float)
                        z = 0.000000 (32 bit float)
                };
        };
        entity_appearance = 0 (32 bit enum)
        entity_marking = {
                char_set = 0x0 (8 bit enum)
                char_strings[11] (8 bit unsigned int) =  0 0 0 0 0 0 0 0 0 0 0
        };
        capabilities = 0x0 (32 bin boolean)
        num_articulate_params = 2 (8 bit unsigned int)
        articulat_params_head = {
                [0] = {
                        change = 10 (16 bit unsigned int)
                        id_attached_to = 11 (16 bit unsigned int)
                        parameter_type = 12 (32 bit enum)
                        parameter_value[8] (8 bit unsigned int) =  31 255 245 200 0 4 24 144
                };
                [1] = {
                        change = 20 (16 bit unsigned int)
                        id_attached_to = 21 (16 bit unsigned int)
                        parameter_type = 22 (32 bit enum)
                        parameter_value[8] (8 bit unsigned int) =  32 255 245 200 239 255 245 128
                };
        };
}
```

APPENDIX B:

SAMPLE PDU LOG FORMATS

INTENTIONALLY LEFT BLANK.

Sample PDU Log Formats

The DIS Network Manager is capable of logging PDUs in one of several formats: straight binary, straight ASCII, or ASCII–binary. Additionally, each PDU may be tagged with a time–stamped header that shows the length of the PDU in bytes, the time it was sent or received, and whether it was incoming or outgoing. Incoming only, outgoing only, or both incoming and outgoing PDUs may be logged.

Below is a sample of an EntityStatePDU logged in two different formats: straight ASCII and ASCII–binary. The values for the fields in the EntityStatePDU may not be correct or valid as this is just sample data used in generating a PDU that could be logged.

Straight ASCII Log Format

Below is an EntityStatePDU logged in the straight ASCII format with a header. The version of the DIS protocol being used is 2.0.3.

```
176 Thu Oct 20 08:07:10.757701 INCOMING
Entity State PDU:
{
        protocol_header = {
                protocol_version = 3 (8 bit unsigned int)
                exercise_ident = 1 (8 bit unsigned int)
                pdutype = 1 (8 bit enum)
                length = 176 (8 bit unsigned int)
                time_stamp = 0 (8 bit unsigned int)
        };
        entity_id = {
                site = 37 (16 bit unsigned int)
                host = 53 (16 bit unsigned int)
                entity_id = 1 (16 bit unsigned int)
        };
        force_id = 2 (8 bit enum)
        entity_type = {
                entity_kind = 0 (8 bit enum)
                domain = 0 (8 bit enum)
                country = 0 (16 bit enum)
                category = 0 (8 bit enum)
                subcategory = 0 (8 bit enum)
                specific = 4 (8 bit enum)
                extra = 0 (8 bit enum)
        };
        alt_entity_type = {
                entity_kind = 0 (8 bit enum)
                domain = 0 (8 bit enum)
                country = 0 (16 bit enum)
                category = 0 (8 bit enum)
                subcategory = 0 (8 bit enum)
                specific = 0 (8 bit enum)
```

```
                extra = 0 (8 bit enum)
};
entity_location = {
                x = 1.000000 (64 bit float)
                y = 2.000000 (64 bit float)
                z = 3.000000 (64 bit float)
};
entity_linear_velocity = {
                x = 0.000000 (32 bit float)
                y = 0.000000 (32 bit float)
                z = 0.000000 (32 bit float)
};
entity_orientation = {
                psi = 0.000000 (32 bit float)
                theta = 0.000000 (32 bit float)
                phi = 0.000000 (32 bit float)
};
dead_reckon_params = {
                algorithm = 0 (8 bit enum)
                linear_accel = {
                                x = 0.000000 (32 bit float)
                                y = 0.000000 (32 bit float)
                                z = 0.000000 (32 bit float)
                };
                angular_velocity = {
                                x = 0.000000 (32 bit float)
                                y = 0.000000 (32 bit float)
                                z = 0.000000 (32 bit float)
                };
};
entity_appearance = 0 (32 bit enum)
entity_marking = {
                char_set = 0x0 (8 bit enum)
                char_strings[11] (8 bit unsigned int) =  0 0 0 0 0 0 0 0 0 0 0
};
capabilities = 0x0 (32 bin boolean)
num_articulate_params = 2 (8 bit unsigned int)
articulat_params_head = {
                [0] = {
                                change = 10 (16 bit unsigned int)
                                id_attached_to = 11 (16 bit unsigned int)
                                parameter_type = 12 (32 bit enum)
                                parameter_value[8] (8 bit unsigned int) =  31 0 0 0 0 0 0 128
                };
                [1] = {
                                change = 20 (16 bit unsigned int)
```

```
                                    id_attached_to = 21 (16 bit unsigned int)
                                    parameter_type = 22 (32 bit enum)
                                    parameter_value[8] (8 bit unsigned int) =  32 1 0 28 16 1 43 32
                            };
                    };
}
```

ASCII–Binary Log Format

Below is the same EntityStatePDU logged in the ASCII–binary format with a header. The version of the DIS protocol being used is 2.0.3. This format prints each bit of the PDU as an ASCII one, "1," or zero, "0."

```
176 Thu Oct 20 08:07:59.037769 INCOMING
00000011000000010000000100000000000000000000000000000000000000000000000000010110000000000000000
00000000000000100101000000000110101000000000000001000000100000001000000000000000000000000000000
00000000000000000000000000001000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000111111111100000000000000000000000000000000000000000000000000000010000000000000
00000000000000000000000000000000000000000000000010000000001000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000010100000000000010110000000000000
00000000000000011000001111100000000000000000000000000000000000000000000000000010000000000000000
00010100000000000001010100000000000000000000000010110001000000000001000000000000111000001
00000000000100101011001000000
```

INTENTIONALLY LEFT BLANK.

APPENDIX C:

DIS_MGR MANUAL PAGE

INTENTIONALLY LEFT BLANK.

NAME

dis_mgr – facility for sending and receiving Distributed Interactive Simulation (DIS) protocol data units (PDU)

SYNOPSIS

dis_mgr [-v] [-n network_interface] [-c num_clients] [-r recv -s send] [-l filename [-a | -b| |-ab] [-i | -in | -incoming] [-o | -out | -outgoing] [-h | -hn] [-ap | -append] [-over | -overwrite]

DESCRIPTION

dis_mgr is a program that provides network communication services to application programs that wish to send and receive DIS PDUs across a network. It uses TCP/UDP for exchanging PDUs on the network. It acts as a server to client application programs maintaining a TCP/IP based connection over which incoming and outgoing PDUs are passed between the dis_mgr and client programs.

The purpose of the dis_mgr is to provide a common service to DIS application programs. That service is to establish and maintain network connections through which DIS PDUs can be sent to and received from other DIS applications. The dis_mgr alleviates application programs from the burden of having to establish and maintain such a network connection themselves. Application programs wishing to use this facility must include the dis_mgr library, libdis. This library contains the function calls through which the services of the dis_mgr are accessed.

OPTIONS

-v	Verbose mode. This option causes the dis_mgr to detail its actions through standard output.
-n net_iface	net_iface specifies the name of the ethernet connection over which PDUs will be broadcast and received, e.g., le0.
-c num_clients	num_clients indicates the maximum number of client application programs that can use the services of the dis_mgr at any one time. Default is 4.
-r recv -s send	recv and send are used to specify alternate UDP port numbers that the dis_mgr will use to send and receive DIS PDUs over the network. This is useful if more than one independent DIS exercise is being conducted simultaneously on the same network.
-l filename	Turns on the logger facility within the dis_mgr. Logged PDUs are written to the file called filename.
-a | -b | -ab	Format of logfile. -a – PDUs are logged in straight ASCII. -b – PDUs are logged in binary. -ab – PDUs are logged in ASCII-binary, that is, the binary representation of each PDU is logged in the file as a series of ASCII one's ("1") and zeros ("0"). This mode is to facilitate debugging when it is necessary to see the actual bit stream of a PDU. Binary is default format when -l option is specified.

-i | -in | -incoming	For logging PDUs, this option indicates that only incoming PDUs are to be logged. On by default when -l option is specified.
-o | -out | -outgoing	For logging PDUs, this option indicates that only outgoing PDUs are to be logged. On by default when -l option is specified.
-h | -hn	PDUs can be logged with a header detailing the time, length, and direction (incoming or outgoing) of the PDU. -h (the default) turns logging with headers on and -hn turns it off.
-ap | -append	Indicates that logged PDUs should be appended to the logfile.
-over | -overwrite	Indicates that logged PDUs should overwrite anything previously existing in the logfile.

SEE ALSO

libdis(1).

APPENDIX D:

LIBDIS MANUAL PAGE

INTENTIONALLY LEFT BLANK.

NAME

libdis – software library of C routines for interfacing to the Distributed Interactive Simulation (DIS) Network Manager program, dis_mgr.

SYNOPSIS

#include <dis_lib.h> - located in src/H under the source directory of the DIS Network Manager distribution.

int **dis_open**(char * host)

int **dis_close**(void)

int **dis_register_pdu**(char * pdu_list)

int **dis_ignore_pdu**(char * pdu_list)

int **dis_send**(char * pdu)

int **dis_read**(char ** pdu)

pdu_type * **get_*PduType*PDU**( )

void **free_*PduType*PDU**(*pdu_type* * pdu)

char * **print_*PduType***(*pdu_type* * pdu)

int **NEW_ENTITY_ID**(void)

int **my_pdu**(char * pdu)

DESCRIPTION

These routines constitute the DIS Network Manager library libdis. libdis is a collection of C routines that implement a TCP/IP based communication interface between an application program and the dis_mgr program. The purpose is to facilitate the exchange of DIS protocol data units (PDU) with other DIS application programs on the network. (See dis_mgr(1).)

dis_open establishes a TCP/IP based network connection between an application program, hereafter referred to as *client*, and a dis_mgr. host is an ASCII character string for the standard network node hostname of the computer on which dis_mgr is running. This routine must be used by a client before any other libdis routines. If successful, it returns a value of TRUE (1), otherwise FALSE (0) is returned. **dis_close** closes or terminates a connection between a client and the dis_mgr.

dis_register_pdu is used by clients to tell the dis_mgr which type of PDUs it wishes to have sent to it. Conversely, **dis_ignore_pdu** is used to specify which types of PDUs are not to be sent to the client. For both of these routines pdu_list is a white space separated list of PDU types (by number). For example, "1 2 5." The PDU types by number are listed in the DIS header file "dis_lib.h." Clients can use these two routines to control which types of incoming PDUs the dis_mgr will send to it. This enables clients to only have to handle PDUs that it wants to process. **dis_register_pdu** and **dis_ignore_pdu** return TRUE (1) if successful and DL_BAD_NETCONN (−2) if a dis_mgr connection has not been established or does not exist for some reason.

dis_send is a routine that clients use to pass a PDU to the dis_mgr for transmission across the network to all other listening DIS applications. The argument, pdu, is a pointer to a C data structure that describes the PDU. This pointer must be typecast as a (char *) before being used in **dis_send**. The "dis_lib.h" header file contains C data structures for every type of PDU that fully correspond to the PDU definitions as described in the proposed IEEE Standard Draft: *Standard for Information Technology - Protocols for Distributed Interactive Simulation Applications, Version 2.0 Drafts 2 and 3*, Institute for Simulation and Training, Orlando, FL. **dis_send** returns DL_BAD_NETCONN (−2) if there is not a good network connection to the dis_mgr. DL_BAD_PDU (−1) is returned if an unrecognized PDU type is sent or if the PDU structure could not be converted into a binary stream for some reason. If successful, **dis_send** returns TRUE (1).

dis_read is a routine used by clients to receive the next incoming PDU that has been forwarded to it by the dis_mgr. The argument, pdu, is to be the address of a pointer so that **dis_read** can set it to point to the place in memory where the next incoming PDU resides. **dis_read** returns an integer that corresponds to the incoming PDU type by number (as listed in "dis_lib.h"), DL_NO_DATA (0) indicating that there is no PDU, or DL_BAD_NETCONN (−2) if the network connection does not exist or is bad. **dis_read** operates on the principle that clients will request the next incoming PDU from the dis_mgr, hence **dis_read** should be used in the main loop of the client program and called as frequently as possible.

A set of routines of the form **get_*PduType*PDU** exist that return a pointer to a dynamically allocated PDU data structure suitable for use by the client program. In this structure, the site and host fields are initialized to their correct values by the **get_*PduType*PDU** routine. In actual use, *pdu_type* should be replaced with the name of a valid PDU type. For example, to dynamically allocate an Entity State PDU structure, use **get_EntityStatePDU**. A corresponding set of routines of the form **free_*PduType*PDU** exist for freeing dynamically allocated memory obtained via the **get_*PduType*PDU** routines.

Similarly, a set of routines of the form **print_*PduType*** exist for printing the contents of a PDU. pdu is a pointer to the C data structure formatted PDU that is to be printed. These routines return a pointer to a new string containing the contents of pdu suitable for use with any of the C string(3C) functions. The returned string must be used or copied before a successive call to the same **print_*PduType*** routine.

NEW_ENTITY_ID is a routine that returns the next available entity number for use in the Entity Identification field of PDUs. It is to be used every time a new entity is created by a client. This function ensures unique entity ID numbers among all clients of a particular dis_mgr. **my_pdu** takes a pointer to a PDU data structure, pdu, cast as a (char *) and determines if it is a PDU that originated from the client using this routine. If so, it returns TRUE (1), otherwise FALSE (0) is returned.

SEE ALSO

dis_mgr(1).

LIST OF ACRONYMS

ACISD	Advanced Computational and Informational Sciences Directorate
ARL	U.S. Army Research Laboratory
DIS	Distributed Interactive Simulation
GUI	Graphical User Interface
ID	Identification
LAN	Local Area Network
PDU	Protocol Data Unit
SMB	Simulation Methodology Branch
TCP/IP	Transport Control Protocol/Internet Protocol
TCP/UDP	Transport Control Protocol/User Datagram Protocol

INTENTIONALLY LEFT BLANK.

NO. OF COPIES	ORGANIZATION
2	ADMINISTRATOR ATTN DTIC DDA DEFENSE TECHNICAL INFO CTR CAMERON STATION ALEXANDRIA VA 22304-6145
1	DIRECTOR ATTN AMSRL OP SD TA US ARMY RESEARCH LAB 2800 POWDER MILL RD ADELPHI MD 20783-1145
3	DIRECTOR ATTN AMSRL OP SD TL US ARMY RESEARCH LAB 2800 POWDER MILL RD ADELPHI MD 20783-1145
1	DIRECTOR ATTN AMSRL OP SD TP US ARMY RESEARCH LAB 2800 POWDER MILL RD ADELPHI MD 20783-1145
	ABERDEEN PROVING GROUND
5	DIR USARL ATTN AMSRL OP AP L (305)

NO. OF COPIES	ORGANIZATION
1	OFFICE OF THE ASSISTANT SECRETARY RESEARCH DEV AND ACQUISITION ATTN GEORGE T SINGLEY III 103 ARMY PENTAGON WASHINGTON DC 20310-0103
1	US ARMY STRICOM PM DIS ATTN AMSTI CSS CENTRAL FLORIDA RESEARCH PARK 12350 RESEARCH PKWY ORLANDO FL 32826-3276
1	COMMANDER ATTN ATZL CDB COL PATRICK LAMAR 415 SHERMAN AVE FT LVNTHWRTH KS 66027-1344
1	COMMANDER ATTN ATCL B COMBAT SVC SUPPORT BATTLE LAB CASCOM FT LEE VA 23801
1	COMMANDER ATTN ATSF CBL USA FIELD ARTILLERY DEPTH & SMLTNUS ATTACK BATTLE LAB FT SILL OK 73503-5600
1	COMMANDANT ATTN ATSH CDA DISMOUNTED BATTLESPACE BATTLE LAB USIS FT BENNING GA 31905-5400
1	COMMANDER ATTN EARLY ENTRY BATTLE LAB ATCD L HQ TRADOC FT MONROE VA 23651-5000
1	COMMANDER ATTN ATZK MW USARMC MOUNTED BATTLESPACE BATTLE LAB FT KNOX KY 40121-5000
1	DIRECTOR ATTN ATRC WSR US ARMY TRADOC ANALYSIS CMD WSMR NM 88002-5502

NO. OF COPIES	ORGANIZATION
2	ARPA ATTN ASTO COL REDDY 3701 N FARFAX DR ARLINGTON VA 22203-1714
3	DIRECTOR ATTN AMSRL HR MT MR MATTHEWS US ARMY RESEARCH LABORATORY 12350 RESEARCH PKWY ORLANDO FL 32826-3276
3	UNIV OF CENTRAL FLORIDA INST FOR SIMULATION & TRAINING 3280 PROGRESS DR ORLANDO FL 32826
1	UNIVERSITY OF NEW MEXICO UNIVERSITY HILL NORTHEAST ALBUQUERQUE NM 87131
2	NAVAL POSTGRADUATE SCHOOL ATTN DAVID R PRATT PHD COMPUTER SCIENCE DEPT MONTEREY CA 93943-5100
1	LORAL FEDERAL SYSTEMS ATTN JOSEPH BRANN 12461 RESEARCH PKWY SUITE 400 ORLANDO FL 32826
1	THE MITRE CORPORATION ATTN JOE LACETERA 145 WYKOFF RD EATONTOWN NJ 07724-0000
1	CAE LINK ATTN CHRISTINA BOUWENS MS 604 PO BOX 1237 BINGHAMTON NY 13902-1237
1	LORAL WESTERN DEV LABS ATTN DAVID GOBUTY 3200 ZANKER RD MS X 38 SAN JOSE CA 95161-0000

NO. OF COPIES	ORGANIZATION
1	MERTEK INC ATTN MICHAEL ROTHROCK 1395 ENTERPRISE OSTEEN RD ENTERPRISE FL 32725-9404
1	SAIC ATTN MARK HOPTIAK 10129 TURNBERRY PLACE MS 282 OAKTON VA 22124
1	STRICOM ATTN AMSTI TD RON HOFER 12350 RESEARCH PKWY ORLANDO FL 32826-3276
2	COMMANDER ATTN AMSRL HR MF LINDA PIERCE ARL HRED USAFAS FT SILL OK 73503-5600
	ABERDEEN PROVING GROUND
26	DIR USARL ATTN: AMSRL-SC, W MERMAGEN, SR COL J BLAKE R K LODER AMSRL-SC-S, A MARK AMSRL-SC-SS, V A KASTE J BOWEN C E HANSEN K T KIRK R J PEARSON T A PURNELL M QIU M A THOMAS W ZHOU AMSRL-SC-SM, K D FICKIE AMSRL-SC-SA, LTC J A WALL MAJ M BIEGA J FORESTER E G HEILMAN V LONG J F O'MAY AMSRL-SC-II, H A INGHAM

NO. OF COPIES	ORGANIZATION
	AMSRL-WT-WE, W P JOHNSON G C SAUERBORN J LACETERA P HILL AMSRL-HR-MF, R SPENCER
2	DIR USAMSAA ATTN: AMSXY-CD, W BROOKS W HUGHES
4	CDR USACSTA ATTN: STECS-AC-FS, A ETZEL C LAMBERT C MARTIN MAJ M SMITH
2	CDR USATECOM ATTN: AMSTE-CT-M, J CHEW J KNOX

INTENTIONALLY LEFT BLANK.

USER EVALUATION SHEET/CHANGE OF ADDRESS

This Laboratory undertakes a continuing effort to improve the quality of the reports it publishes. Your comments/answers to the items/questions below will aid us in our efforts.

1. ARL Report Number ARL-TR-780 Date of Report June 1995

2. Date Report Received _____

3. Does this report satisfy a need? (Comment on purpose, related project, or other area of interest for which the report will be used.) _____

_____

_____

4. Specifically, how is the report being used? (Information source, design data, procedure, source of ideas, etc.)

_____

_____

_____

5. Has the information in this report led to any quantitative savings as far as man-hours or dollars saved, operating costs avoided, or efficiencies achieved, etc? If so, please elaborate. _____

_____

_____

6. General Comments. What do you think should be changed to improve future reports? (Indicate changes to organization, technical content, format, etc.) _____

_____

_____

_____

CURRENT ADDRESS

_____
Organization

_____
Name

_____
Street or P.O. Box No.

_____
City, State, Zip Code

7. If indicating a Change of Address or Address Correction, please provide the Current or Correct address above and the Old or Incorrect address below.

OLD ADDRESS

_____
Organization

_____
Name

_____
Street or P.O. Box No.

_____
City, State, Zip Code

(Remove this sheet, fold as indicated, tape closed, and mail.)
(DO NOT STAPLE)

DEPARTMENT OF THE ARMY

OFFICIAL BUSINESS

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO 0001,APG,MD

POSTAGE WILL BE PAID BY ADDRESSEE

DIRECTOR
U.S. ARMY RESEARCH LABORATORY
ATTN: AMSRL-SC-SS
ABERDEEN PROVING GROUND, MD 21005-5067